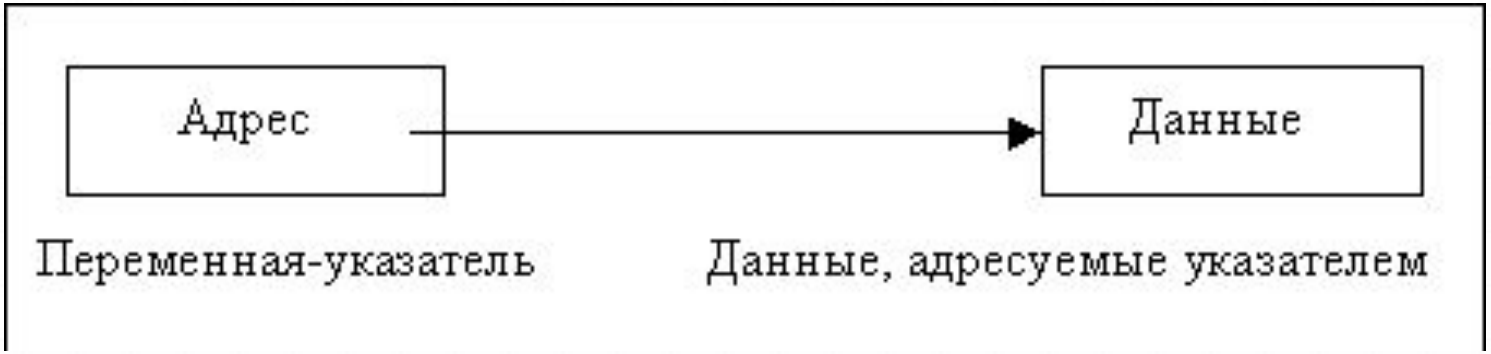
& *

Указатели позволяют

- обрабатывать многомерные и одномерные массивы, строки, символы, структуры и массивы структур.
- динамически создавать новые переменные в процессе выполнения программы.
- обрабатывать связанные структуры: стеки, очереди, списки, деревья, сети.
- передавать функциям адреса фактических параметров.
- передавать функциям адреса функция в качестве параметров.

Определение

- **Указатель**-это переменная или константа, которая содержит значение адреса другой переменной.



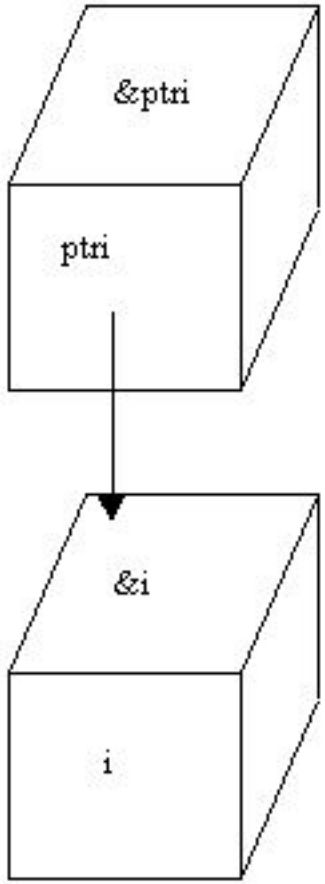
Объявление указателей и основные операции над ними

- **тип [модификатор] *<имя-указателя>**
- **тип**-имя типа переменной, адрес которой будет содержать переменная- указатель.(например integer, char, long)
- **имя-указателя** –идентификатор переменной типа указатель.(имя собственное)
- *- определяет переменную типа указатель.

- Значение переменной-указателя-это адрес некоторой величины, целое без знака.
- Указатель содержит адрес первого байта переменной определённого типа.
- Тип адресуемой переменной, на которую ссылается указатель, определяет объём оперативной памяти, выделяемой переменной, связанной с указателем.

- указатель содержит адрес нулевого байта этой переменной
- тип адресуемой переменной определяет, сколько байтов, начиная с нулевого (адреса, определённого указателем) занимает это значение

Примеры объявлений указателей



&ptri	-адрес указателя.
ptri=&i;	-значение указателя, адрес i.
&i	-адрес переменной i.
i==*ptri;	-значение переменной.

& и *

- &-получение адреса переменной.
- *-извлечение значения, расположенного по этому адресу.

&-имя переменной

- получение адреса, **определяет адрес** размещения значения переменной определённого типа. Операндом операции **&** должно быть имя переменной того же типа, для которой определён и указатель левой части оператора присваивания, получающий значение этого адреса.

*-имя указателя

- получение значения определённого типа по заданному адресу. Определяет содержимое, находящееся по адресу, который содержится в указателе-переменной или указателе-константе. Иначе: **косвенная адресация**.

Косвенная адресация

- помощью операции * осуществляет доступ к значению по указателю, то есть извлечение значения, расположенного по адресу-содержимому указателя. Операнд * (т.е имя после) должно быть типа указатель (ранее объявлено).

Инициализация указателя

- **имя указателя_переменной=&имя_переменной**
- **int *ptri,i;**
- //объявление указателя и переменной типа int
- **ptri=&i;**
- //ptri получает значение адреса ‘i’

- оператор присваивания, использующий имя указателя и * операцию косвенной адресации:
- **Имя_переменной=*имя_указателя**
- Имя указателя –это переменная или константа, которая содержит адрес размещаемого значения, требуемого для переменной левой части оператора присваивания

- **i=*ptri;**
- // ‘i’ получает значение, расположенное по адресу
- // содержащемся в указателе ‘ptri’

Взаимосвязь указателя, адреса и значения переменной

```
int *pi          //указатель-переменная на данные типа integer
char *pc         //указатель-переменная на данные типа char
float *pf        //указатель-переменная на данные типа float
int m1[5]        //объявление массива m1 типа int на 5 значений
                 //m1-указатель-константа
int *m2[10]      //m2-имя массива на 10 значений типа указатель на
                 //значения типа int
```

Указатели можно использовать

- ***ptri-**значение переменной, находящейся по адресу, содержащемуся в указателе ptri
- **ptri-**значение адреса переменной
- **&ptri-**адрес местоположения самого указателя

- int i=123, j, *ptri;
- //объявление переменных и указателя
- ptri=&i;
- //инициализация указателя(присвоение адреса i)
- j=*ptri+1;
- //переменной i (*ptri) присваивается значение
- //переменной i и к её содержимому прибавляется единичка.

Многоуровневая адресация

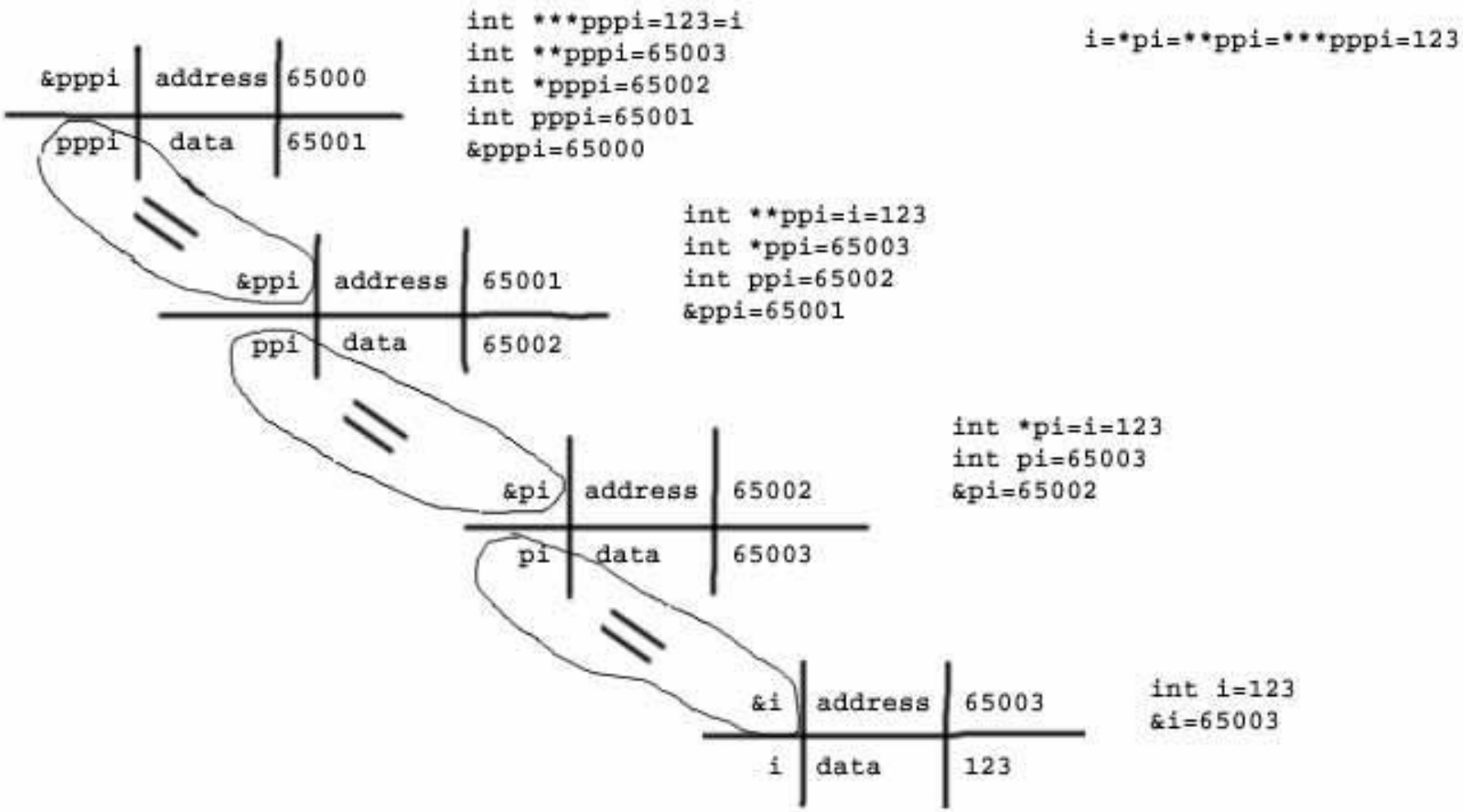
- **int i=123;**
- //где i-имя переменной
- **int *pi=&i;**
- //pi –указатель на переменную
- **int **ppi=π**
- //ppi-указатель на ‘указатель на переменную’
- **int ***pppi=&ppi;**
- //pppi-указатель на ‘указатель на ‘указатель на переменную”.

Правила

- Полное количество звёздочек косвенной адресации, равное количеству звёздочек при объявлении указателя, определяет значение переменной.
- Уменьшение количества звёздочек косвенной адресации добавляет к имени переменной слово ‘**указатель**’, причём этих слов может быть столько, сколько может быть уровней косвенной адресации для этих имён указателей, то есть столько, сколько звёздочек стоит в объявлении указателя.

Соответствие между количеством уточнений (*) и результатом обращения к значению с помощью указателя

Обращение	Результат обращения	У.К.А.
i	Значение переменной i	0
*pi	Значение переменной, на которую указывает pi	1
pi	Указатель на переменную типа int, знпчение pi	0
**ppi	Значение переменной типа int	2
*ppi	Указатель на переменную типа int	1
ppi	Указатель на ‘указатель на переменную типа int’, значение указателя ppi	0
***pppi	Значение переменной i типа int	3
**pppi	Указатель на переменную типа int	2
*pppi	Указатель на ‘указатель на переменную типа int’	1
pppi	Указатель на ‘указатель на ‘указатель на переменную типа int’’,значение указателя pppi	0


int ***pppi=123=i
int **pppi=65003
int *pppi=65002
int pppi=65001
&pppi=65000
i=*pi=**ppi=***pppi=123
&pppi address 65000
pppi data 65001
int **ppi=i=123
int *ppi=65003
int ppi=65002
&ppi=65001
&ppi address 65001
ppi data 65002
int *pi=i=123
int pi=65003
&pi=65002
&pi address 65002
pi data 65003
int i=123
&i=65003
&i address 65003
i data 123

Операции над указателями

- Присвоить указателю значение адреса данных, или нуль.
- Увеличить (уменьшить) значение указателя
- Прибавить (вычесть) из значения указателя целое число
- Сложить или вычесть значение одного указателя из другого
- Сравнить два указателя с помощью операций отношения.

задать значение Переменной-указателю

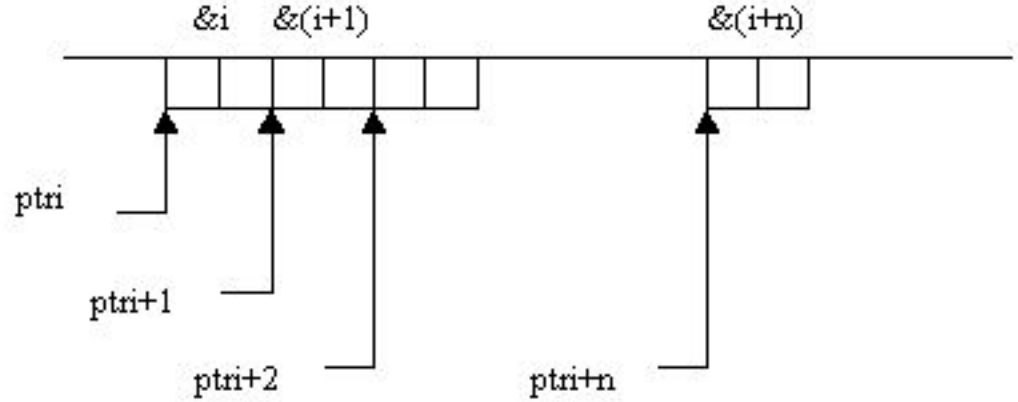
- Присвоить указателю адрес переменной, имеющей место в оперативной памяти, или нуль:
- **ptri=&i;**
- **ptri=NULL;**
- Объявить указатель вне функции (в том числе main) либо в любой функции, снабдив его описателем **stastic,** при этом начальным значением указателя является нулевой адрес (**NULL**)

- Присвоить указателю значение другого указателя, который к этому времени уже инициализирован (имеет определённое значение), например: **ptri=ptrj;** -это двойное указание одной и той же переменной.
- Присвоить переменной-указателю значение с помощью функций **malloc** и **calloc.**

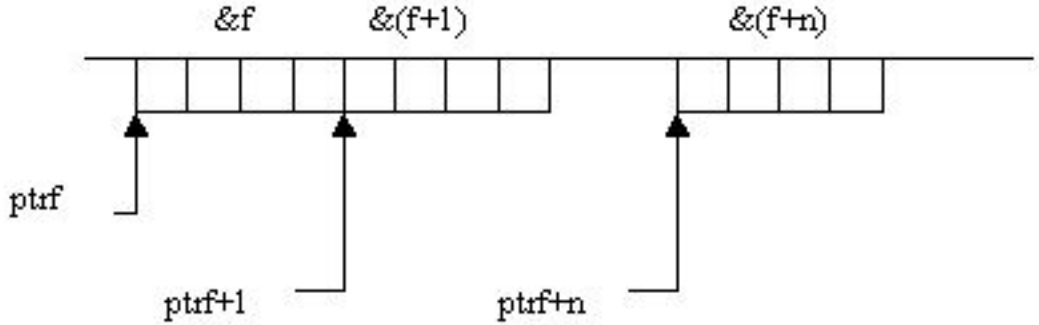
Изменение значений указателя

- +,
- ++,
- -,
- --

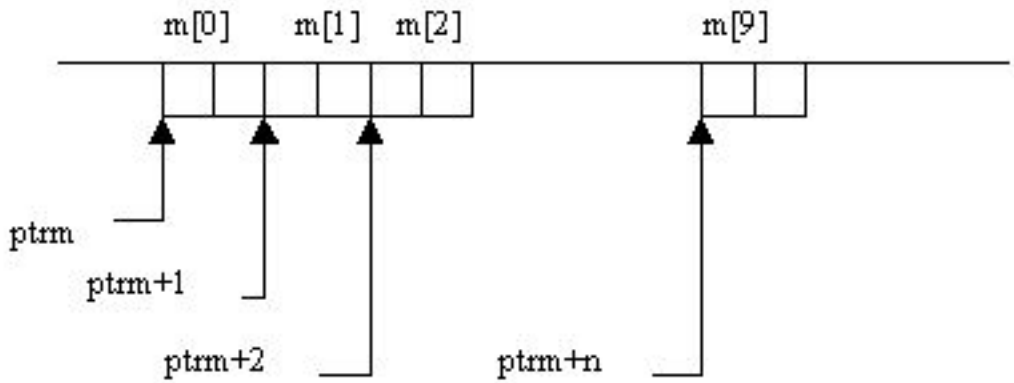
```
int i,*ptri;
ptri=&i;
ptri++;
```




```
float f,*ptrf;
ptrf=&f;
ptrf++;
```




```
int m[10],*ptrm;
ptrm=&m[0];
ptrm++;
```



Связь с массивами

- int mas[10],*ptrm;
- ptrm=&mas[0];
- *prtm==mas[0]==*(mas+0) ;
- //значение нулевого элемента массива mas
- *(ptrm+i)==mas[i]==*(mas+i);
- //значение i-го элемента массива mas

- *mas+2==mas[0]+2;
- *(mas+i)-3==mas[i]-3;

*(&(mas[i+1])+2)++;

- ptrm==&mas[i+1];
- //упрощение выражения, i не играет роли
- ptrm+2==&(mas[i+1])+2;
- //указатель переводится на 2 элемента вперёд
- ***ptrm++==**(*ptrm=*ptrm+1);
- //содержимое ячейки массива извлекается и к нему прибавляется единичка

префиксные (слева от имени указателя) **постфиксные** (справа от имени указателя)

- Префиксные операции в последовательности справа налево.
- Использование значения, полученного после выполнения префиксных операций
- Постфиксная операция над указателем.

- ***p++** сначала выполняется префиксная операция над указателем ,то есть определяется значение ***p**- содержимое, расположенное по адресу **px**, а затем выполняется постфиксная операция **++** увеличение значения указателя на квант памяти, то есть на 2 байта (если указатель типа int)

0 **(++(*p)+2)** сначала:

- *p -так как префиксные операции выполняются справа налево.
- *p=*p+1 -самая ‘левая’ префиксная операция
- **+2 -**выполнение постфиксной операции

Проблемы

- Попытка работать с неинициализированными указателям, то есть с указателем, не содержащим адреса оперативной памяти (ОП), выделенной переменной.
- Потеря ссылки, то есть значения указателя из-за присваивания ему нового значения до освобождения ОП, которою он адресует.
- Не освобождение ОП, запрошенной с помощью функции **malloc**

Синтаксис указателей

- data_type *pointerName;
- data_type — тип данных,
- pointerName — имя указателя.

- int *integerPointer;

- // Объявление указателя и простой переменной в одной строке
- int *pointer1, // это указатель
- variable; // это обычная переменная типа int
-
- // Объявление двух указателей в одно строке
- int *pointer1, // это указатель с именем pointer1
- *pointer2; // это указатель с именем pointer2

два способа использования указателя

1. Использовать имя указателя без символа *, таким образом можно получить фактический адрес ячейки памяти, куда ссылается указатель.
2. Использовать имя указателя с символом *, это позволит получить значение, хранящееся в памяти. В рамках указателей, у символа * есть техническое название — операция разыменования. По сути, мы принимаем ссылку на какой-то адрес памяти, чтобы получить фактическое значение.

Объявление указателя, получение адреса переменной

- Для того чтобы объявить указатель, который будет ссылаться на переменную,
- необходимо сначала получить адрес этой переменной.
- Чтобы получить адрес памяти переменной (её расположение в памяти), нужно использовать знак & перед именем переменной.
- Это позволяет узнать адрес ячейки памяти, в которой хранится значение переменной.
- Эта операция называется — операция взятия адреса

- int var = 5;
- // простое объявление переменной с предварительной инициализацией
- int *ptrVar;
- // объявили указатель, однако он пока ни на что не указывает
- ptrVar = &var;
- // теперь наш указатель ссылается на адрес в памяти, где хранится число 5

```
#include <stdio.h>

int main()
{
    int var;     // обычная целочисленная переменная
    int *ptrVar; // целочисленный указатель (ptrVar должен быть типа int, так как он будет ссылаться на переменную типа int)

    ptrVar = &var;       // присвоили указателю адрес ячейки в памяти, где лежит значение переменной var
    scanf( "%d", &var );  // в переменную var положили значение, введенное с клавиатуры
    printf( "%d\n", *ptrVar ); // вывод значения через указатель
    getchar();
}
```

- http://cppstudio.com/post/9088/
- http://cppstudio.com/post/423/
- http://cppstudio.com/post/429/